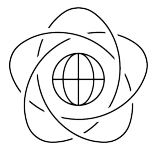
Programming 3 Advanced

Xml documentation, Assembly, Disposable

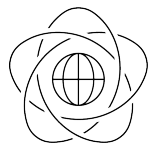
Tomasz Herman

Faculty of Mathematics and Information Science
Warsaw University of Technology

Lecture 9, 4 grudnia 2024



Outline



XML documentation

```
1 /// <summary>
2 /// Abstract base class representing a mathematical matrix.
3 /// </summary>
4 public abstract class Matrix;
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/recommended-tags

Examples

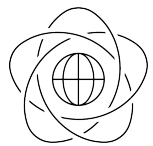
https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/examples

XML escape characters

There are only 5 characters to escape:

```
1  "    &quot;
2  '    &apos;
3  <    &lt;
4  >    &gt;
5  &    &amp;
```

```
1  /// <summary>
2  /// This property always returns a value &lt; 1.
3  /// </summary>
```



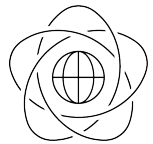
General tags

Summary:

```
1 /// <summary>
2 /// description
3 /// </summary>
```

Additional remarks:

```
1 /// <remarks>
2 /// description
3 /// </remarks>
```



Describing method

Return value:

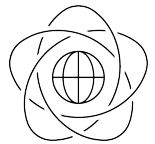
```
<returns>description</returns>
```

Parameter:

```
<param name="name">description</param>
```

Exception:

```
<exception cref="ExceptionType">description</exception>
```

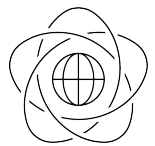


Formating

para tag

Paragraph:

```
<remarks>
    <para>
        This is an introductory paragraph.
    </para>
    <para>
        This paragraph contains more details.
    </para>
</remarks>
```

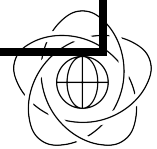


Formating

list tag

List (or table):

```
1   <list type="bullet|number|table">
2     <listheader>
3       <term>term</term>
4       <description>description</description>
5     </listheader>
6    <item>
7       <term>Assembly</term>
8       <description>
9         The library or executable built from a compilation.
10      </description>
11    </item>
12  </list>
```



Examples

Inline code:

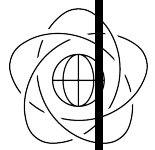
```
<c>text</c>
```

Multiple lines of code:

```
<code>
    var index = 5;
    index++;
</code>
```

Example:

```
<example>
This shows how to increment an integer.
<code>
    var index = 5;
    index++;
</code>
</example>
```



Resusing documentation

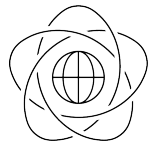
Inheriting the documentation:

```
1 <inheritdoc [cref=""] [path=""]/>
```

- by default inherits the documentation from base class or interface

From an xml file:

```
1 <include file='filename' path='tagpath[@name="id"]' />
```



Links and references

Inline cross-reference:

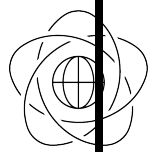
```
<see cref="member"/>
<!-- or -->
<see cref="member">Link text</see>
<!-- or -->
<see href="link">Link Text</see>
<!-- or -->
<see langword="keyword"/>
```

cref = code reference

href = hyperlink reference (url)

Cross-reference:

```
<seealso cref="member"/>
<!-- or -->
<seealso href="link">Link Text</seealso>
```



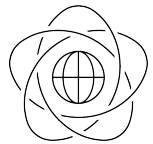
Generic types and methods

Generic parameter:

```
1 <typeparam name="TResult">description</typeparam>
```

Referencing generic parameters:

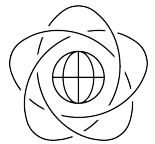
```
1 <see cref="Foo{T,U}"/>
```



Generating xml documentation

In .csproj file:

```
<PropertyGroup>
    <DocumentationFile>doc.xml</DocumentationFile>
</PropertyGroup>
```

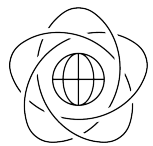


Generating HTML, pdf or chm documentation

Sandcastle `https://github.com/EWSoftware/SHFB`
- Used to generate MSDN documentation
- Designed to work with C# , F# , and Visual Basic
- Can build HTML, pdf and chm documentation

DocFX `https://dotnet.github.io/docfx/`
- Primarily designed to work with .NET languages
- Language support extendable through plugins
- Can build html or pdf documentation

Doxygen `https://github.com/doxygen/doxygen`
- Supports many languages: C, C++ , C# , java, python, and many more
- Can build html, LaTeX documentation or manpages.



Assembly

Assembly

An assembly is the basic unit of deployment in .NET. Assemblies take the form of executable (.exe) or dynamic link library (.dll) files.

Assembly contains:

- Assembly manifest
- Application manifest
- Compiled types
- Resources

Documentation

`https://learn.microsoft.com/en-us/dotnet/standard/assembly/`

The Assembly Manifest

Manifest

It describes the assembly to the managed hosting environment and acts as a directory to the types, and resources in the assembly.

Assembly manifest contains:

- Name and version of the assembly
- List of types defined in the assembly
- List of resources in the assembly
- List of referenced assemblies

Documentation

`https://learn.microsoft.com/en-us/dotnet/standard/assembly/manifest`

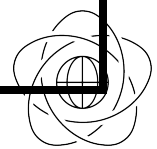
Embedded Resources

Adding a file as an embedded resource in .csproj

```
<ItemGroup>
    <None Remove="file.txt" />
    <EmbeddedResource Include="file.txt" />
</ItemGroup>
```

Getting a stream to an embedded file

```
Assembly a = Assembly.GetEntryAssembly();
using (Stream s = a.GetManifestResourceStream
    ("Project.file.txt"))
{
    // ...
}
```



Localized embedded resources

```
1  ResourceManager rm = new ResourceManager
2      ("EmbeddedResources.Resources", typeof(Program).Assembly);
3  string? welcome = rm.GetString("Welcome",
4      CultureInfo.CreateSpecificCulture("pl"));
5  string? demo = rm.GetString("Demo");
6  Console.WriteLine(welcome);
7  Console.WriteLine(demo);
```

Documentation

Creating resources

https://learn.microsoft.com/en-us/dotnet/core/extensions/create-resource-files

Retrieving resources

https://learn.microsoft.com/en-us/dotnet/core/extensions/retrieve-resources

Assembly resources

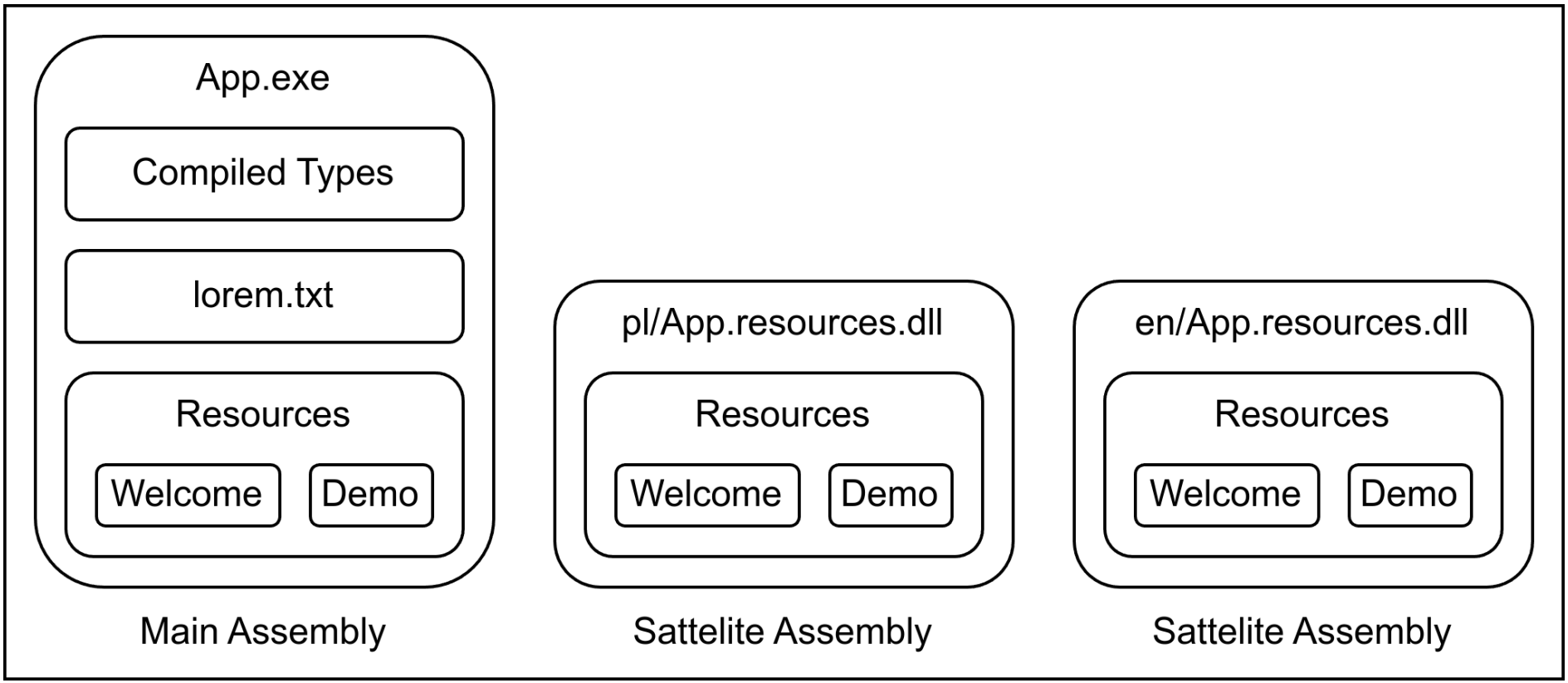
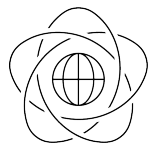


Figure: An assembly that contains one embedded file lorem.txt and resources container Resources.resx with two localized strings Welcome and Demo



Assembly Loading Context

Every .NET and .NET Core application implicitly uses AssemblyLoadContext. It's the runtime's provider for locating and loading dependencies. Whenever a dependency is loaded, an AssemblyLoadContext instance is invoked to locate it.

- AssemblyLoadContext provides a service of locating, loading, and caching managed assemblies and other dependencies.
- To support dynamic code loading and unloading, it creates an isolated context for loading code and its dependencies in their own AssemblyLoadContext instance.

Documentation

`https://learn.microsoft.com/en-us/dotnet/core/dependency-loading/understanding-assemblyloadcontext`

Assembly Loading Context

Getting ALC for specific Assembly

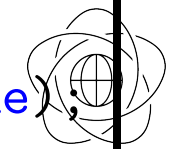
```
1  Assembly assem = Assembly.GetExecutingAssembly();
2  AssemblyLoadContext context =
3      AssemblyLoadContext.GetLoadContext(assem);
4  Console.WriteLine(context.Name);
5
6  foreach (Assembly a in context.Assemblies)
7      Console.WriteLine(a.FullName);
```

Getting the default ALC

```
1  AssemblyLoadContext alc = AssemblyLoadContext.Default;
```

Creating a new ALC

```
1  AssemblyLoadContext context =
2      new AssemblyLoadContext("Plugins", isCollectible: true);
```

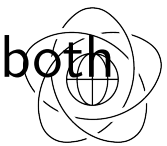


Loading Assemblies into ALC

```
1 AssemblyLoadContext alc = AssemblyLoadContext.Default;
2 Assembly assembly = context
3     .LoadFromAssemblyPath(assemblyPath);
```

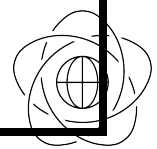
Assembly resolution

1. The CLR first checks whether an identical resolution has already taken place in that ALC (with a matching full assembly name); if so, it returns the Assembly it returned before.
2. Otherwise, it calls the ALC's (virtual protected) Load method, which does the work of locating and loading the assembly. With a custom ALC, it's entirely up to you how you locate the assembly.
3. If Step 2 returns null, the CLR then calls the Load method on the default ALC
4. If Step 3 returns null, the CLR then fires the Resolving events on both ALCs - first, on the default ALC and then on the original ALC.



Custom ALC

```
1  class PluginLoadContext : AssemblyLoadContext
2  {
3      private AssemblyDependencyResolver _resolver;
4
5      public PluginLoadContext(string pluginPath, bool collectible = true)
6          : base(name: pluginPath, isCollectible: collectible)
7      {
8          _resolver = new AssemblyDependencyResolver(pluginPath);
9      }
10
11     protected override Assembly? Load(AssemblyName assemblyName)
12     {
13         string? assemblyPath = _resolver
14             .ResolveAssemblyToPath(assemblyName);
15         if (assemblyPath != null)
16         {
17             return LoadFromAssemblyPath(assemblyPath);
18         }
19         return null;
20     }
21 }
```



IDisposable interface

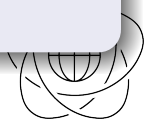
IDisposable interface

IDisposable interface provides a mechanism for releasing unmanaged resources.

```
1 public interface IDisposable
2 {
3     void Dispose();
4 }
```

Documentation

https://learn.microsoft.com/en-us/dotnet/fundamentals/runtime-libraries/system-idisposable



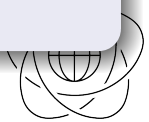
using statement

```
1 using (var fs = new FileStream("file.txt", FileMode.Open))
2 {
3     // ...
4 } // Automatically calls Dispose() method on fs
```

```
1 using var fs = new FileStream ("file.txt", FileMode.Open);
2 // ...
3 // Automatically calls Dispose() when fs goes out of scope
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/using

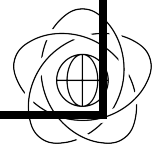


using statement

```
1  using (var fs = new FileStream("file.txt", FileMode.Open))
2  {
3      // ...
4  } // Automatically calls Dispose() method on fs
```

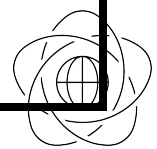
The compiler converts this to the following:

```
1  var fs = new FileStream ("file.txt", FileMode.Open);
2  try
3  {
4      // ...
5  }
6  finally
7  {
8      if (fs != null) ((IDisposable)fs).Dispose();
9  }
```



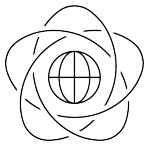
Implementing IDisposable

```
1  public class Logger : IDisposable
2  {
3      private readonly StreamWriter sw;
4
5      public Logger(string path)
6      {
7          sw = new StreamWriter(path, append: true);
8      }
9
10     public void Log(string message) => sw.WriteLine(message);
11
12     public void Dispose()
13     {
14         sw.Dispose();
15     }
16 }
```



Dispose semantics

- After calling `Dispose`, calling methods or other properties should throw `ObjectDisposedException`
- Calling `Dispose` method repeatedly causes no error
- Disposable object should also dispose all disposable objects it owns



Calling Dispose from finalizer

```
1  class Test : IDisposable
2  {
3      public void Dispose()
4      {
5          Dispose(true);
6          GC.SuppressFinalize(this); // Prevents running finalizer
7      }
8
9      protected virtual void Dispose(bool disposing)
10     {
11         if (disposing)
12         {
13             // Call Dispose() on objects owned by this instance.
14         }
15         // Release unmanaged resources owned by just this object.
16     }
17
18     ~Test() => Dispose(false);
19 }
```